Breaking and Fixing the Self Encryption Scheme for Data Security in Mobile Devices

Paolo Gasti
DISI - University of Genoa
Via Dodecaneso 35, 16146 Genova, Italy
Email: gasti@disi.unige.it

Yu Chen
State University of New York - Binghamton
Binghamton, NY 13902, USA
Email: ychen@binghamton.edu

Abstract—**Data security is one of the major challenges that prevents the wider acceptance of mobile devices, especially within business and government environments. It is non-trivial to protect private and sensitive data stored in these devices due to the limited resources and computing power, particularly when they fall in the hand of an adversary. Previously Chen and Ku proposed a lightweight data encryption and storage scheme named Self-Encryption (SE) to meet the challenge. However, our recent research revealed that there are critical weaknesses in SE. This paper presents the detailed analysis of the weaknesses of SE scheme and proposes a solution to remove the flaws in SE. Through real-world measurements on top of the iPhone platform, we verified the effectiveness of our proposal.**

Index Terms—**mobile computing, pervasive computing, data security, Self-Encryption, stream ciphers, smartphone, PDA, iPhone.**

I. Introduction

There is little doubt that Pervasive Computing [1] is becoming an integral part of everyday life. The social consequences of this development are profound [2], and the technological challenges posed by this paradigm are complex and interesting (see for example [3], [4]). One of these challenges is maintaining data privacy for the users. Portability makes mobile devices prone to be stolen or lost. These devices usually carry sensitive business or private information: government employees store classified documents on their mobile devices [5]; corporate users save confidential files, PINs and passwords on their devices; users keep photos, phone numbers, personal appointments and other private information in their PDAs or smartphones.

As a consequence, the content of such devices should be protected from unauthorized access. However it is desirable that this protection does not introduce a substantial computational overhead, which would reduce both the performance of mobile devices and their battery life – and therefore their usefulness.

The Self-Encryption (SE) Scheme for Data Security in Mobile Devices was introduced by Chen and Ku in 2009 [6]: in their paper they have investigated wether it is possible to implement a lightweight encryption algorithm which provides data confidentiality by exploiting the availability of a secure connection with a central server. The server is used to store a small amount of data, required to decrypt the confidential information. In case of loss of the device, the access to the company server is temporarily revoked.

Unfortunately, we discovered that SE possesses some critical weaknesses. In this paper we illustrate our discoveries about the vulnerabilities of the scheme. We also suggest a way to take advantage of the ideas introduced with SE using standard tools in order to provide an efficient and strongly secure encryption scheme. We show that this approach can be a viable solution in processing-power and battery constrained environments through some real-world measurements.

Organization of the paper: In the next section we illustrate how mobile devices vendors currently protect the data stored in mobile devices. In Section III we show why SE is not secure and in Section IV we show how the ideas behind SE can be used to construct a secure and efficient scheme. We conclude in Section V.

II. Related work

There are currently many different approaches embraced by different vendors in order to protect mobile users from data theft. Corporate oriented platforms, such as the E series from Nokia, Windows Mobile from Microsoft and Blackberry from RIM offer a features called remote wipe [7]; more recently, the same feature is starting to appear on consumer oriented platforms, such as the iPhone from Apple and the Pre from Palm. The remote wipe feature is simple and, at first glance, seems also effective: in case of loss of the device, the owner can issue a specific command, which tells the device to erase all the stored data. Once its content has been securely erased, the device is completely useless for an adversary who is interested in obtaining confidential information.

A skilled adversary, however, can easily overcome this kind of protection by simply applying standard forensics procedures: she can remove, when possible, the memory card containing the sensitive data, she can remove the battery or she can use a faraday shielding bag to stop the device from communicating with the network. Therefore, we consider the remote wipe approach appropriate only when the target of the theft is the devices itself, and not the information it contains.

Most devices provide encryption tools in order to protect the information contained in their memory. There are currently two approaches: encrypting the whole content of the device or providing a small encrypted storage area for confidential information. We believe that encrypting the whole content of a mobile device may not be a good choice: compared to a desktop or laptop processor, an embedded CPU is computationally much weaker. Encrypting the whole content of the device introduces unnecessary delays every time the system accesses to a non-confidential file. Performing a decryption (and, in some cases, also an encryption) at every access of the flash memory can seriously reduce the battery life [8], making this option unattractive for the end user. Moreover, desktop operating systems mandate the use of whole disk encryption even to protect just a small subset of sensitive data, because usually the operating system's metadata can reveal information – and often parts of the content – about sensitive files [9]. This threat does not seem to be as relevant in mobile devices. Many devices provide a small encrypted storage space to store confidential information. This functionality, often referred to as "mobile wallet" [10], takes care of storing user's passwords and other confidential information such as short notes or credit card numbers. However this storage space does not support arbitrary file formats. In addition, both full-disk encryption and "mobile wallet" base their security on the strength of the password chosen for the encryption. A short password would make brute force attacks feasible, and would therefore render the protection ineffective. On the other hand, choosing a long and complex password would force the users to struggle with space constrained input devices, such as touch screens or small keyboards.

A. The Self-Encryption Scheme

The aim of SE is to efficiently encrypt selected sensitive files requiring the user to enter just a short PIN, while achieving the same security as using a high-entropy password. In order to achieve this goal, SE exploits the availability of a secure remote server which can store some secret information. SE does not require any TPM device and can be easily implemented as a user-space application on most smartphone. Unfortunately, we discovered that the encryption algorithm used in SE does not guarantee data privacy.

Since the main purpose of this paper is to give a detailed analysis of the weaknesses of the Self-Encryption Scheme, we would like to give a detailed description of the scheme itself. Therefore, we define below the probabilistic polynomial-time algorithms for parameter generation (*Gen*), decryption (*Dec*) and encryption (*Enc*) which compose the Self-Encryption Scheme.

Let G be a pseudorandom generator and H a collision-resistant hash function. The Self-Encryption Scheme is defined as follows:

***Gen*:** on input 1^n, chose $k \leftarrow \{0,1\}^n$ uniformly at random and output it as the PIN.

***Enc*:** takes in input a PIN $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$, $\Delta \in [0,1]$, 1^l and a security level $S_L \in \mathbb{N}$. Then it encrypts the message m as follows:

- choose a nonce $v \leftarrow \{0,1\}^l$ and set $s = H(k,v)$
- if $S_L > 0$, set $j = |m| \cdot S_L \cdot \Delta$. Otherwise set $j = 256$
- set R as the first $j \cdot \log_2(|m|)$ bits of the output of $G(s)$
- split R in j values $r_0, \ldots, r_{j-1}$ such that every value r_i has a length of $\log_2(|m|)$ bits
- for $i = 0$ to j, set $r_i = r_i \bmod (|m| - i)$
- construct a keystream $ks \in \{0,1\}^j$ where the i-th bit of ks is equal to the r_i-th bit of m and mark each bit r_i on the message
- construct a plaintext string $pl \in \{0,1\}^{|m|-j}$ by inserting the bit of the message that were not marked in the previous step
- output ks, v and the ciphertext c as $c = pl \oplus ks$.

***Dec*:** takes in input a PIN k, a ciphertext c, a keystream ks, Δ, S_L and a nonce v. Then it decrypts the ciphertext c as follows:

- set $s = H(k,v)$
- calculate $pl = c \oplus ks$
- if $S_L > 0$, set $j = |m| \cdot S_L \cdot \Delta$. Otherwise set $j = 256$
- set R as the first $j \cdot \log_2(|m|)$ bits of the output of $G(s)$
- split R in j values $r_0, \ldots, r_{j-1}$ such that every value r_i has a length of $\log_2(|m|)$ bits
- for $i = 0$ to j, set $r_i = r_i \bmod (|m| - i)$
- output $m \in \{0,1\}^{|ks|+|pl|}$ where, for $i = 0$ to j the r_j-th bit of m is equal to the i-th bit of ks and where the remaining bits of m are set as the bits in pl.

The keystream ks is saved on the secure server after the encryption, while v, c, S_L and Δ are stored on the device. ks must be retrieved from the secure server before running the decryption algorithm.

In the security analysis we model the secure server used to store the keystream as completely impenetrable by not providing the adversary the information that the secure server protects.

III. Breaking the Self-Encryption Scheme

In this section we recall first the notion of chosen plaintext attack (CPA). [6]. Next, we show that the scheme is not secure since it succumbs against a chosen plaintext attack and other simple real-world attacks.

A. IND-CPA Security

The weakest reasonable security notion commonly accepted for a practical encryption scheme is IND-CPA security. This property is often defined through a game played by a challenger and an adversary $\mathcal{A}$. The basic idea behind the IND-CPA game is that $\mathcal{A}$ is allowed to ask for encryptions of multiple messages chosen adaptively and then choose a pair of messages m_0 and m_1 and have the challenger to encrypt one of them chosen randomly. The adversary wins if she can correctly guess which message was encrypted by the challenger. Formally, the standard IND-CPA game is defined as follows:

Setup: The challenger runs $Setup(1^\kappa)$ to generate the encryption key k.

Phase 1: The adversary $\mathcal{A}$ obtains encryptions of any messages $m \in \mathcal{M}_k$ of its choice, where $\mathcal{M}_k$ is the message space for the encryption algorithm Enc under key k.
Challenge: $\mathcal{A}$ specifies two challenge messages $m_0, m_1 \in \mathcal{M}_k$ and sends them to the challenger. The challenger chooses a random bit $b \in \{0,1\}$. If $b = 0$, it returns $Enc_k(m_0)$ to $\mathcal{A}$; otherwise, it returns $Enc_k(m_1)$ to $\mathcal{A}$.
Phase 2: The adversary can obtain more encryptions of messages from the message space.
Guess: The adversary outputs a bit $b' \in \{0,1\}$ as its guess for b. $\mathcal{A}$ wins the game if $b' = b$.
The advantage of adversary $\mathcal{A}$ winning the above game is defined as $Adv_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}} = |\Pr[b' = b] - 1/2|$.

An encryption scheme $\mathcal{E} = (Setup, Enc, Dec)$ is said to have indistinguishable encryptions the under chosen plaintext attack if for every polynomial p, polynomial-time adversary $\mathcal{A}$, and all sufficiently large κ it holds that $Adv_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}} < \frac{1}{p(\kappa)}$.

B. The Attack

The Self-Encryption scheme presented above is not secure against a chosen plaintext attack. Let $\mathcal{A}$ be an adversary who plays the chosen plaintext attack game against the challenger. $\mathcal{A}$ skips **Phase 1** of the IND-CPA game and selects two same-length messages m_0 and m_1 for the **Challenge** phase, where m_0 is a random bit string and m_1 is a string in which every bit is set to 1. Then $\mathcal{A}$ sends m_0 and m_1 to the challenger. The challenger chooses a random bit $b \in \{0,1\}$ and returns the ciphertext for the encryption of m_b to $\mathcal{A}$.

If $b = 1$ then the ciphertext is the encryption of m_1: the challenger picks a pseudo-randomly chosen subset of the bits of m_1 to form the keystream ks. Since every bit of m_1 is set to 1, all the possible keystreams have all their bits set to 1. Note that also the bits that were not chosen for the keystream, and therefore composing the plaintext pl, are all set to 1. As a consequence, no matter which PIN, nonce, security level or Δ is chosen by the adversary, all the bits of the ciphertext obtained by XORing pl with ks are always set to 0.

Instead, if $b = 0$, the ciphertext will look completely random to the adversary since both the keystream and the plaintext are two random bit strings.

Therefore, $\mathcal{A}$ outputs "$b' = 1$" if all the bits in the ciphertext are set to 0, and "$b' = 0$" otherwise. $\mathcal{A}$ outputs a value $b' = b$ with very high probability, more specifically

$$\Pr[b' = b] = 1 - \mathsf{negl}(\ |\ c\ |\ )$$

where $\mathsf{negl}(\cdot)$ is a negligible function. For this reason, SE does not have indistinguishable encryption under chosen plaintext attack.

Note that the proposed attack works in constant time in the length of the PIN, the size of the nonce and the value S_L: the adversary distinguishes between encryptions of m_0 and encryptions of m_1 by simply matching the ciphertext with a known string, i.e. $0^{|c|}$.

C. Statistical Bias and Full Plaintext Recovery

In the previous section we showed that SE is not CPA-secure. At first, the attack against the CPA security of SE may seem unrealistic and its effect may be underestimate, especially if we keep into account the security/speed tradeoff that was considered when designing SE. However, we found that SE is vulnerable to even more devastating attacks, which are discussed here. More specifically, in this section we show that the bits in the ciphertext c are usually not uniformly distributed in $\{0,1\}$, and that it is easy to mount a ciphertext-only attack, i.e. retrieve meaningful information about the underlying message by just observing the ciphertext.

The keystream is generated by uniformly choosing j bits from the message m, therefore the bits in the keystream tend to follow the same distribution as the bits in the message m. Let us denote with pl the plaintext, with c the ciphertext, with ks the keystream and with $str[i]$ the i-th bit of a bit string str. We have that

$$\Pr\big[c[i] = 0\big] = \Pr\big[pl[i] = 1\big] \cdot \Pr\big[ks[i] = 1\big] + \Pr\big[pl[i] = 0\big] \cdot \Pr\big[ks[i] = 0\big]$$

Whenever the bits of the message are non-negligibly biased towards a specific value, the bits of the ciphertext have a non-negligible bias as well. Therefore the encryption scheme lacks the property of diffusion [11]. A uniformly distributed keystream and a uniformly distributed output are a fundamental requirement for any secure stream cipher [12].

Another serious problem with this scheme is that the keystream can be shorter than the plaintext. In this case, the keystream must be re-used within the same encryption. This has devastating consequences, since it fails in hiding the structure of the message [11] and can lead to a recovery of the plaintext by just observing the ciphertext [12]. Let us give a simple example of this: let m be an XML file containing passwords and other personal information, such as credit card numbers. Since the information contained in this file are very sensitive, the user encrypts m with SE, producing the ciphertext c. Usually password manager software saves, together with the password, also the username, URL, a title for the entry, creation, access and modification dates and some notes. Thanks to this and the verbosity of the XML language [13], a password file can easily surpass 128KB in size. For simplicity we set $S_L = 0$ and the size of m to 128KB. The keystream size is therefore 256 bits, which means that 256 bits are randomly extracted from the message. XML files usually start with the same preamble, i.e. `<?xml version="1.0" encoding="UTF-8"?>`. This string, which is of course known to adversary, has a length of 38 bytes which correspond to 304 bits[1]. An adversary can calculate the XOR of the first 256 bits of c with the first 256 bits of the XML preamble,

[1]The known preamble is usually longer than 304 bits, since the structure of the XML file is known as well and therefore the adversary can correctly predict which XML tag is the root and wich tags are its first children. In the case of the KeePassX [14] XML format, for example, the adversary knows the first 464 bits of the password file.

obtaining the whole keystream. The keystream obtained is correct if none of the first 256 bits of m were chosen for the keystream. In the example above, the adversary obtains the correct keystream with a probability of about 94%. If some of the first 256 bits of m have been chosen for the keystream, the adversary can still reconstruct it as soon as at least 256 out of the 304 known bits are not chosen. This happens with overwhelming probability in the example above.

After obtaining the keystream, the adversary calculates the plaintext by XORing the keystream with the ciphertext. Thanks to the redundancy of XML files, it is very easy to efficiently reconstruct m, i.e. re-inserting the keystream bits in the appropriate positions, from just the keystream and the plaintext without knowing the PIN. We point out that increasing the size of S_L does not prevent this kind of attack, unless half of the bits composing the XML file or more is used as keystream. However, in this case, the adversary has a lower success probability (but still non negligible) and longer running time; the attack is still well into the realm of feasibility.

Message integrity is another issue that wasn't addressed by the original SE paper. We believe that SE should provide data integrity, since it protects from a class of attack which are very realistic with mobile devices: imagine that an adversary can modify an encrypted message in a way that this modification is not detectable by the user. If this adversary can access the device, even for a limited time, she can alter the content of arbitrary sensitive files. The user will not notice this attack, and will trust the content of such files. Since SE does not provide any method for message authentication, when an adversary flips a bit of the ciphertext, the corresponding bit in the plaintext will be flipped. The adversary can usually guess the position of a specific plaintext bit in the ciphertext with very high probability.

Due to the weaknesses exposed so far, we strongly recommend to avoid the use of SE to protect any sensitive information.

IV. Fixing the Self-Encryption Scheme

We think that the core idea of the Self-Encryption scheme, i.e. exploiting the availability of a secure server not accessible to the adversary in order to strengthen the encryption, is brilliant and can solve some of the problems which currently affect mobile device security.

Mobile devices are feature-crippled if compared to their desktop counterpart. Processing power and battery capacity limitations do not allow the execution of complex tasks, and the limits imposed by their input interfaces are well known. However, the advances in embedded processors and the availability of fast stream ciphers, such as RC4 [15], Rabbit [16] or Salsa20 [17], make the use of such ciphers viable on resource constrained platforms. We believe that such ciphers can be used as an efficient building block for a strongly secure encryption scheme, and we detail one of the possible constructions in the following sections.

A. The RC4 Stream Cipher

Stream ciphers are well known for their high speed and small memory footprint compared to block ciphers. Our interest goes towards RC4, a small pseudo-random generator, due to its fame and widespread availability.

RC4 was introduced in 1987 by Ron Rivest [15]. Today, among other uses, it is one of the available ciphers for SSL/TLS [18] (secure socket layer / transport layer security). It is composed of a key scheduling phase, where the secret key – typically between 40 and 256 bits long – is used to initialize an internal state. The stream sequence is then produced by outputting specific values from the internal state, which is updated after each byte of output is produced. Unlike many other stream ciphers, such as those in eStream [19], RC4 does not take an initialization vector together with the key. Therefore each implementation must specify how to combine the initialization vector and the key in order to generate each time a different RC4 keystream. A survey of some of the approaches to combine the key and the initialization vector is given in [20].

RC4 is widely considered secure when used appropriately. Specific care must be invested in order to make sure that the pseudorandom generator is used properly, since "plain" RC4 is now known to have some weaknesses. For one, the first few bytes of its output are known to be biased [21]. Therefore, when RC4 is used, the first 128 or so bytes of its output should be discarded. Together with this weakness, there are resynchronization problems (see for example [22] and [23]). We assume that the implementation of a systems which uses RC4 takes the appropriate countermeasures against known weaknesses.

B. Ciphers Performances on Mobile Devices

In order to determine the performance impact of encrypting documents on the device using a standard cipher, we measured the speed of some widely used stream and block ciphers on a modern smartphone.

Our testbed is the Apple iPhone 3G. Released in 2008, it is an interesting device, since it is a full Unix machine in a very small package. It is powered by a 416MHz Samsung 32-bit RISC CPU, which is a modern low-powered in-order CPU based on the ARM instruction set, rated 0.45 mW/MHz (with cache), with 16KB of level 1 instruction cache and 16KB of level 1 data cache [24]. The iPhone 3G comes with 128MB of on-board RAM. ARM based CPUs are currently the most widely used CPUs in mobile devices, accounting for approximately 90% of all embedded 32-bit RISC processors. Similarly clocked or even faster ARM-based processors can be found on many Nokia E and N series smart phones, Microsoft Zune, Nokia N800 and N810 tablet, Motorola RIZR, most Windows Mobile smartphones, Palm Pre, Samsung Omnia, Blackberry devices, and Google's Android-based devices, just to cite some.

We were able to install OpenSSL [25] on the iPhone in order to perform some tests. OpenSSL is a well known open source library which provides efficient implementations of many symmetric and asymmetric cryptographic algorithms, together with various utility functions. Since the only stream cipher available with OpenSSL is RC4, we were unable to compare it with other stream ciphers like Rabbit or Salsa20 at this time. However, an implementation which uses OpenSSL as a cryptographic library would necessarily have to use RC4, therefore we think that our measurements are meaningful. We compared RC4 with the AES and DES block ciphers using a 128-bit and a 56-bit key respectively, in order to provide reference values. The performance figures given below are obtained using OpenSSL 0.9.8k.

TABLE I
PERFORMANCES MEASUREMENTS ON THE TEST PLATFORM. THE RESULTS ARE EXPRESSED IN KILOBYTES PER SECOND FOR THE SPECIFIED PLAINTEXT SIZE. THE BLOCK CIPHERS ARE USED IN CBC MODE.

	64 Bytes	256 Bytes	1 KB	8 KB
RC4	19,847	20,438	21,294	20,934
AES-128	5,303	5,504	5,740	5,611
DES	3,492	3,559	3,681	3,623
Triple DES	1,269	1,324	1,266	1,285

Among the ciphers provided by OpenSSL, RC4 is clearly the best solution for a mobile device: it provides adequate performances for most tasks and outperforms the implementations of all the tested block ciphers. This translates into low latencies when accessing encrypted files and lower power consumption – and therefore longer battery life – compared to block ciphers.

The iPhone 3G also provides UMTS [26] connectivity up to 3.6 Mbps and WiFi 802.11b/g up to 54 Mbps. UMTS connectivity is provided by the Infineon S-Gold3H [27] chip. The power consumption of this chip in UMTS mode is rated about ten times higher than the power consumption of the main processor, therefore in our solution we try reduce the data exchanged with the secure server to a minimum.

C. SE as a Key Management Tool

In order to fix SE's weaknesses, we briefly outline a way to exploit a secure remote server as a key management device. We focus primarily on two important aspects: i) users, especially mobile users, clearly prefere to remember and type a short PIN instead of a long complex password, and ii) mobile devices usually have limited input interface. This makes long and complex passwords – such us the ones including symbols and punctuation – difficult to type.

No matter how strong the encryption algorithm is, short and simple passwords are the weakest link when protecting encrypted sensitive files. Therefore we exploit the remote secure server to store part of the secret information associated with each encrypted file. The key used to encrypt a specific document is generated from this secret information and a user PIN. The user must be authenticated to the secure server in order to retrieve this secret. We suggest to minimized the interaction with the secure server because, as we have shown before, transmitting and receiving information through a wireless network is the major source of power consumption and is much slower than decrypting a ciphertext with a fast cipher, e.g. RC4.

Similarly to SE, here we want to allow a user to choose which files are stored as encrypted in the devices storage space. The algorithm we propose to encrypt local files taking advantage of a remote secure storage is as follows: after selecting a file to encrypt, the user specifies a PIN. The device then selects a random secret s, an initialization vector IV and a random file identifier ID. The lengths of s, IV and ID must be appropriate, e.g. 128 bits for s and IV and 80 bits for ID. In this way we can guarantee that the secret is hard to guess for an adversary, that the file identifier is unique for a specific user and that a particular value for the initialization vector is not used more than once.

The encryption key k for the selected file is generated as $k = H(s \mid\mid PIN)$, where $H : \{0,1\}^* \rightarrow \{0,1\}^{|s|}$ is a collision resistant hash function. The file is encrypted using the RC4 stream cipher with key k and initialization vector IV and saved on the device, together with the file identifier ID and IV. The secret s and the file identifier are sent to the secure server through a secure and authenticated connection and then deleted from the device. In this way the secure server can associate s with the file identifier, and can efficiently respond to subsequent authenticated requests from the mobile device.

When a user wants to decrypt a specific file, she enters the PIN and logs in through a secure and authenticated connection to the company server. Then she requests the secret corresponding to the encrypted file's ID. The server responds with s, which is then used together with the PIN to reconstruct the key k. The secret s and the PIN are immediately removed from the main memory of the device. A copy of the key k is temporarily saved in the device in order to prevent the need for further requests to the server in case the user wants to modify and re-encrypt the confidential document – in which case a new initialization vector is also chosen. After closing the file, the device deletes the unencrypted version from its memory, together with the key k. The use of a different secret for each encrypted file reduces the information exposure in case the mobile device is snatched from the hands of a user while she is viewing a sensitive document. In this case the adversary can see the file while it is in its unencrypted form. However, the adversary will not be able to obtain, even knowing a specific k, any meaningful information about the other encrypted files, their encryption keys or the PIN, since the secrets saved on the secure server are chosen independently at random for each file.

Mobile devices need to interact with the secure server in order to be able to decrypt confidential files. Therefore access policies can be enforced by the server administrator. As an example, the administrator can prevent all users from accessing sensitive documents outside office hours, or disallow a specific user from viewing confidential files when she's on vacation.

Together with confidentiality, the scheme should provide integrity by associating a message authentication code [12]. The key for the message authentication code can be generated by applying a deterministic function to the key used for encryption and the PIN, such as hashing them together.

D. Network Related Considerations

Exploiting the availability of a remote secure server requires a secure channel between the mobile device and the secure server. Recovering the messages exchanged between the mobile device and the secure server would render the secure server useless: the adversary would be able to gather most of the information needed to decrypt the files on the device, and will just need to guess the PIN – which is supposed to be easier to guess than the secret.

Great care must be used to authenticate both the server and the mobile devices. Due to the wireless nature of the connection of mobile devices, the adversary may have complete control over the network used by the device. The adversary can therefore alter messages between the device and the secure server at her will. As an example, a man in the middle attack can be a realistic threat if the right countermeasures are not taken. Therefore the connection between the device and the server must be protected using an encrypted *and* authenticated channel.

There are many well known and well studied solutions for this problem. The device can be connected to the company network through a Virtual Private Network (VPN), which takes care of providing the required authenticated and encrypted channel. One protocol that is commonly used to build a VPN is IPsec [28], [29]. However, as pointed out in [30], great care must be taken when implementing an IPsec based VPN in order to provide authentication. Most modern smart phones – such as Windows Mobile-based phones, Google's Android phones and the iPhone – already provide support for VPNs based on IPsec.

Another way to build a secure and authenticated connection from the device to the remote server is to use an end to end SSL/TLS [18] connection. The main advantage of this approach is that all the "Internet enabled" smartphones already support SSL and TLS protocols natively.

In order to be able to open an encrypted file, the mobile device must be able to connect with the secure server. The availability of such server becomes therefore crucial, as well as the availability of a network connection to the server. The techniques to provide high availability and ubiquitous connectivity are out of the scope of this paper.

V. Conclusions

It is impending to develop a security scheme to keep confidentiality and integrity of sensitive/private data in the network-based distributed data storage applications. However, it is non-trivial to design a robust solution.

In this paper, we studied the Self-Encryption Scheme, and revealed that although the main idea and the framework of SE are promising, the scheme does not guarantee data confidentiality. To prove this, we gave an example where an adversary can successfully recover the content of an encrypted file without the knowledge of any secret information.

Then we proposed some ideas in order to construct a scheme which can successfully address data privacy problems. We verified the feasibility of our ideas by making a performance analysis which involves experiments on a popular mobile platform.

Acknowledgements

The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century," *Computer*, vol. 36, pp. 25–31, Mar 2003.

[2] M. Arikawa, S. Konomi, and K. Ohnishi, "Navitime: Supporting Pedestrian Navigation in the Real World," *Pervasive Computing, IEEE*, vol. 6, pp. 21–29, July-Sept. 2007.

[3] D. Saha, A. Mukherjee, and S. Bandyopadhyay, *Networking Infrastructure for Pervasive Computing: Enabling Technologies and Systems*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[4] J. Cornwell, I. Fette, G. Hsieh, M. Prabaker, J. Rao, K. Tang, K. Vaniea, L. Bauer, L. Cranor, J. Hong, B. McLaren, M. Reiter, and N. Sadeh, "User-controllable security and privacy for pervasive computing," in *Mobile Computing Systems and Applications, 2007. HotMobile 2007. Eighth IEEE Workshop on*, pp. 14–19, March 2007.

[5] Department of the Army, *Policy on Use of Government Cellular Telephones and Personal Digital Assistant (PDA)*.

[6] Y. Chen and W.-S. Ku, "Self-Encryption Scheme for Data Security in Mobile Devices," in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pp. 1–5, Jan. 2009.

[7] G. S. Punja and R. Mislan, "Mobile device analysis," *Small Scale Digital Device Forensic Journal*, vol. 2, June 2008.

[8] MP3 Insider: The truth about your battery life, `http://tinyurl.com/yzfw6xg`.

[9] A. Czeskis, D. J. S. Hilaire, K. Koscher, S. D. Gribble, T. Kohno, and B. Schneier, "Defeating Encrypted and Deniable File Systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications," in *HOTSEC'08: Proceedings of the 3rd conference on Hot topics in security*, (Berkeley, CA, USA), pp. 1–7, USENIX Association, 2008.

[10] Nokia Press Release: Nokia to introduce a mobile wallet application with the Nokia 6310, `http://tinyurl.com/yk3tc3k`.

[11] C. E. Shannon, "Communication Theory of Secrecy Systems," *Bell Systems Technical Journal*, vol. 28, pp. 656–715, 1949.

[12] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[13] C. J. Augeri, D. A. Bulutoglu, B. E. Mullins, R. O. Baldwin, and L. C. Baird, III, "An analysis of XML compression efficiency," in *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*, (New York, NY, USA), p. 7, ACM, 2007.

[14] KeePassX, a Cross Platform Password Manager, `http://www.keepassx.org/`.

[15] R. Rivest, *The RC4 Algorithm*. RSA Data Security.

[16] M. Boesgaard, M. Vesterager, and E. Zenner, "The rabbit stream cipher," pp. 69–83, 2008.

[17] D. J. Bernstein, "Salsa20 design."

[18] E. Rescorla, *Ssl and Tls*. Boston: Addison-Wesley, 2001.

[19] Network of Excellence in Cryptology ECRYPT. Call for stream cipher primitives, `http://www.ecrypt.eu.org/stream/`.

[20] E. Zenner, "Why IV Setup for Stream Ciphers is Difficult," in *Symmetric Cryptography*, no. 07021 in Dagstuhl Seminar Proceedings, (Dagstuhl, Germany), Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[21] I. Mantin and A. Shamir, "A Practical Attack on Broadcast RC4," in *Proc. of FSE'01*, pp. 152–164, Springer-Verlag, 2001.

[22] I. Mantin, "A Practical Attack on the Fixed RC4 in the WEP Mode," in *ASIACRYPT 2005*, pp. 395–411, 2005.

[23] I. Mantin, "Predicting and Distinguishing Attacks on RC4 Keystream Generator," in *EUROCRYPT 2005*, pp. 491–506, 2005.

[24] The ARM ARM1136J(F)-S CPU Technical Specifications, `http://tinyurl.com/oaaj6b`.

[25] OpenSSL, `http://www.openssl.org`.

[26] K. Richardson, “Umts overview,” *Journal of Electronics and Communication Engineering*, 2000.

[27] Infineon X-Gold 608/XMM 6080 datasheet, `http://tinyurl.com/ylyzfww`.

[28] N. Doraswamy and D. Harkins, *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[29] S. Frankel, K. Kent, R. Lewkowski, A. D. Orebaugh, R. W. Ritchey, and S. R. Sharma, “Guide to IPsec VPNs,” *NIST Special Publication 800-77 (draft)*, January 2005.

[30] K. G. Paterson and A. K. L. Yau, “Cryptography in Theory and Practice: The Case of Encryption in IPsec,” in *EUROCRYPT 2006*, pp. 12–29, 2006.